Here, I am describing the Backend part of **warehouse management**

Produce Warehouse Management System

Project Overview

The Produce Warehouse Management System is a Flask-based web application designed to manage the operations of a direct-to-consumer (D2C) produce warehouse. This system includes features to handle the reception, storage, management, and traceability of produce, ensuring compliance with supply chain traceability and bio-degradable packaging standards. The application is structured to support multiple temperature zones and predict the shelf life of produce items.

The Produce Warehouse Management System is a Flask-based web application designed to manage the operations of a direct-to-consumer (D2C) produce warehouse. This system includes features to handle the reception, storage, management, and traceability of produce, ensuring compliance with supply chain traceability and bio-degradable packaging standards. The application is structured to support multiple temperature zones and predict the shelf life of produce items.

1. Project Structure

The project is organized in a modular structure to ensure maintainability and scalability:

The project is organized in a modular structure to ensure maintainability and scalability:

arduino

arduino

```
produce_warehouse/
```

```
produce_warehouse/
├── app/
│   ├── __init__.py
│   ├── models.py
│   ├── routes.py
│   ├── forms.py
│   ├── templates/
│   │   ├── base.html
│   │   ├── index.html
│   │   └── inventory.html
│   └── static/
│       └── style.css
├── config.py
├── run.py
└── requirements.txt
```

Key Components:

Key Components:

app/: Contains the core application code, including initialization, models, routes, forms, templates, and static files.

app/: Contains the core application code, including initialization, models, routes, forms, templates, and static files.

config.py: Holds configuration settings, such as database connections and secret keys.

run.py: The main entry point for running the application.

requirements.txt: Lists the dependencies required to run the application.

2. Core Functionality

2. Core Functionality

2.1 Receiving Produce

The application allows the warehouse to receive produce shipments. The received produce is stored in a database with details such as name, source, quantity, storage zone, packaging type, and shelf life.

The application allows the warehouse to receive produce shipments. The received produce is stored in a database with details such as name, source, quantity, storage zone, packaging type, and shelf life.

API Endpoint:
API Endpoint:

Route: /api/receive
Route: /api/receive
Method: POST
Input: JSON data with produce details.
Response: Confirms successful receipt and logs the transaction.

2.2 Inventory Management
2.2 Inventory Management

The system provides an overview of the current inventory, listing all produce stored in the warehouse.
The system provides an overview of the current inventory, listing all produce stored in the warehouse.

API Endpoint:
API Endpoint:

Route: /inventory
Route: /inventory
Method: GET
Output: Renders an HTML page with inventory details, including produce name, source, quantity, storage zone, and shelf life.

2.3 Storage Zone Management
2.3 Storage Zone Management

The storage zone for each produce item can be updated to reflect changes in warehouse organization.
The storage zone for each produce item can be updated to reflect changes in warehouse organization.

API Endpoint:
API Endpoint:

Route: /api/storage/<produce_id>
Route: /api/storage/<produce_id>
Method: PUT
Input: JSON data with the new storage zone.
Response: Confirms successful update and logs the change.

2.4 Traceability Logs
2.4 Traceability Logs

The system keeps track of every action taken on the produce, ensuring full traceability.
The system keeps track of every action taken on the produce, ensuring full traceability.

API Endpoint:
API Endpoint:

Route: /api/traceability/<produce_id>
Route: /api/traceability/<produce_id>
Method: GET
Output: JSON data with the produce’s traceability log, including timestamps and actions.

2.5 Shelf-Life Prediction
2.5 Shelf-Life Prediction

The application predicts the remaining shelf life of produce items based on their received date and shelf-life duration.
The application predicts the remaining shelf life of produce items based on their received date and shelf-life duration.

API Endpoint:
API Endpoint:

Route: /api/shelf-life/<produce_id>

Route: /api/shelf-life/<produce_id>
Method: GET
Output: JSON data indicating the remaining shelf life and expiration status.

3. Technical Details

3. Technical Details

3.1 Flask Framework

The application is built using Flask, a lightweight and versatile Python web framework that facilitates rapid development and scalability. Flask's modular design allows for the separation of concerns through blueprints, making the application easy to extend.

The application is built using Flask, a lightweight and versatile Python web framework that facilitates rapid development and scalability. Flask's modular design allows for the separation of concerns through blueprints, making the application easy to extend.

3.2 Database Management

The application uses SQL Alchemy for Object-Relational Mapping (ORM) and SQLite as the database. This setup allows for easy database operations management, including migrations and schema management through Flask-Migrate.

The application uses SQL Alchemy for Object-Relational Mapping (ORM) and SQLite as the database. This setup allows for easy database operations management, including migrations and schema management through Flask-Migrate.

Database Models:

Database Models:

Produce: Represents produce items, including details like name, source, quantity, and shelf life.

Produce: Represents produce items, including details like name, source, quantity, and shelf life.

Traceability Log: Logs every action taken on a produced item, ensuring full traceability.

3.3 Frontend Components

3.3 Frontend Components

The front end is built using HTML, CSS, and Bootstrap for a responsive design. Flask's templating engine, Jinja2, dynamically generates HTML pages based on server-side data.

The front end is built using HTML, CSS, and Bootstrap for a responsive design. Flask's templating engine, Jinja2, dynamically generates HTML pages based on server-side data.

Templates:

Templates:

base.html: The base template for the application, providing a consistent layout.

base.html: The base template for the application, providing a consistent layout.

index.html: The homepage introduces the system's capabilities.
inventory.html: Displays the current inventory with all produce items listed.

Static Files:

Static Files:

style.css: Contains custom CSS to style the application, ensuring a clean and modern user interface.

style.css: Contains custom CSS to style the application, ensuring a clean and modern user interface.